



Tutorial

Prototyping IoT devices on GNU/Linux

*Embedded Linux Conference
#LFELC, Berlin, Germany <2016-10-13>*

Philippe Coval
Samsung Open Source Group / SRUK
philippe.coval@osg.samsung.com

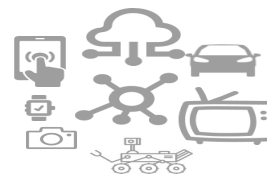
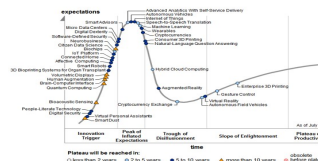
Hallo Welt!

- Philippe Coval
 - Software engineer for Samsung OSG
 - Belongs to SRUK team, based in Rennes, France
 - Ask me for IoTivity support on **Tizen** platform and others
 - Interests
 - Libre Soft/Hard/ware, Communities, Interoperability
 - **DIY**, Embedded, Mobile, Wearables, Automotive...
 - Find me online
 - <https://wiki.tizen.org/wiki/User:Pcoval>



Newbies, makers, hackers welcome !

- This “IoT” talk is not about:
 - Market share, prospects, growth, figures
 - Monetize data with cloud, analytics, big data, machine learning
 - Security, privacy, trust, Skynet, singularity or any concerns
 - Architectures, services or designs
 - Comparison of protocols or implementations
 - Tizen the “OS of Everything” (unless if asked)
- It's about **quick prototyping** for proof of concepts:
 - Learn by doing from scratch, DIY: software, hardware, electronics
 - Feedback on previous **experimentations** from embedded developer
 - Front door to a project of 435K+ lines of code and ~500 pages of specifications



Agenda

- Prototyping
- Simplest example
- Implementation
- Hardware integration
- Demonstration
- Q&A

Motivations for prototyping

- *NOT* making a mass produced IoT device at 1st shot
 - Low cost (<10 \$), low consumption (mW), high level of security
- Validate concepts with **relaxed constraints**
 - In **friendly environment** (ie: tools, security or connectivity shortcuts)
 - Validate, show, gather feedback, stress, benchmark, adapt, iterate
- Think of use cases first?
 - Or experiment with what can technology can provide? Be inspired!
- Topics and Ideas?
 - Controlling, monitoring, convergence, network of sensors, behaviors, AI...



“Simplicity
is the ultimate sophistication.”
~Leonardo da Vinci

Simplest use case

- From the blinking led
- To a remote controlled switch
 - GPIO, LED, Relay, Motor, Fan, Home Appliance...
 - Simple functions: On/Off
- To a flip/flop relay controlled by multiple clients
 - Notification of change in real time
 - Consistent toggle feature
- Identified problems, are half solved :
 - **Sharing** hardware resource(s) through a seamless **connectivity**



IoTivity : Connectivity between devices

- Apache-2 licensed C/C++ Implementation
 - Of Open Connectivity Foundation's **standard** (OCF~OIC)
- Many features:
 - **Discovery** (IETF RFC7252 / IP Multicast)
 - Communication (RESTfull API on CoAP) w/ Security (DTLS)
 - Transports (IP, WiFi, BT, BLE, Zigbee...)
 - Data/Device management, web services, cloud, plugins...
- Today we'll use only few features to connect our thing



OCF Vocabulary is all about resources

- Resource is representing
 - virtual object (ie: logical states)
 - physical data (ie: actuator, sensors)
 - hybrid (ie: soft sensors)
- Resource entity
 - Each can be accessed by an URI
 - Has a resource type identifier
 - Is composed of **properties**
 - type, name, value
- More concepts
 - Model to describe
 - Resource's **interface**
 - Properties & allowed ops
 - GET, POST, PUT, params...
 - Groups, collections, links
 - Scenes, Things manager
 - Many more services

Don't reinvent the wheel

- OCF's Standardized data **model** repository
 - <http://www.oneiota.org/>
 - **RESTful** API Modeling Language (RAML > JSON)
 - To be used with a simulator (ATM)
- Search for existing models
 - <https://github.com/OpenInterConnect/IoTDataModels>
 - <http://www.oneiota.org/documents?q=switch>
 - binarySwitch.raml includes oic.r.switch.binary.json
 - <http://www.oneiota.org/revisions/1580>

OCF Model defines switch resource type

```
{ "id": "http://openinterconnect.org/iotdatamodels/schemas/oic.r.switch.binary.json#",  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "description" : "Copyright (c) 2016 Open Connectivity Foundation, Inc. All rights reserved.",  
  "title": "Binary Switch",  
  "definitions": {
```

```
    "oic.r.switch.binary": {
```

```
      "type": "object",
```

```
      "properties": {
```

```
        "value": {
```

```
          "type": "boolean",
```



```
          "description": "Status of the switch"
```

```
        } } } } // ...
```



“The secret of getting ahead
is getting **started.**”
~ *Mark Twain*

Time to make choice

- OS? <https://wiki.iotivity.org/os>
 - None: for Microcomputers (MCU: Bare metal)
 - **GNU/Linux**  : Debian/Ubuntu, Yocto, Tizen, OpenWRT...
 - Or others FLOSS or not
- Hardware? <https://wiki.iotivity.org/hardware>
 - Arduino (MCU) : C API
 - Cheap **Single Board Computer** (CPU): C++ API (or C API too)
 - IO: GPIO, I2C, SPI, Antennas, Daughter-boards...
 - RaspberryPI (0|1|2|3), MinnowMax (OSHW), Edison,  **ARTIK** (5|10), ...

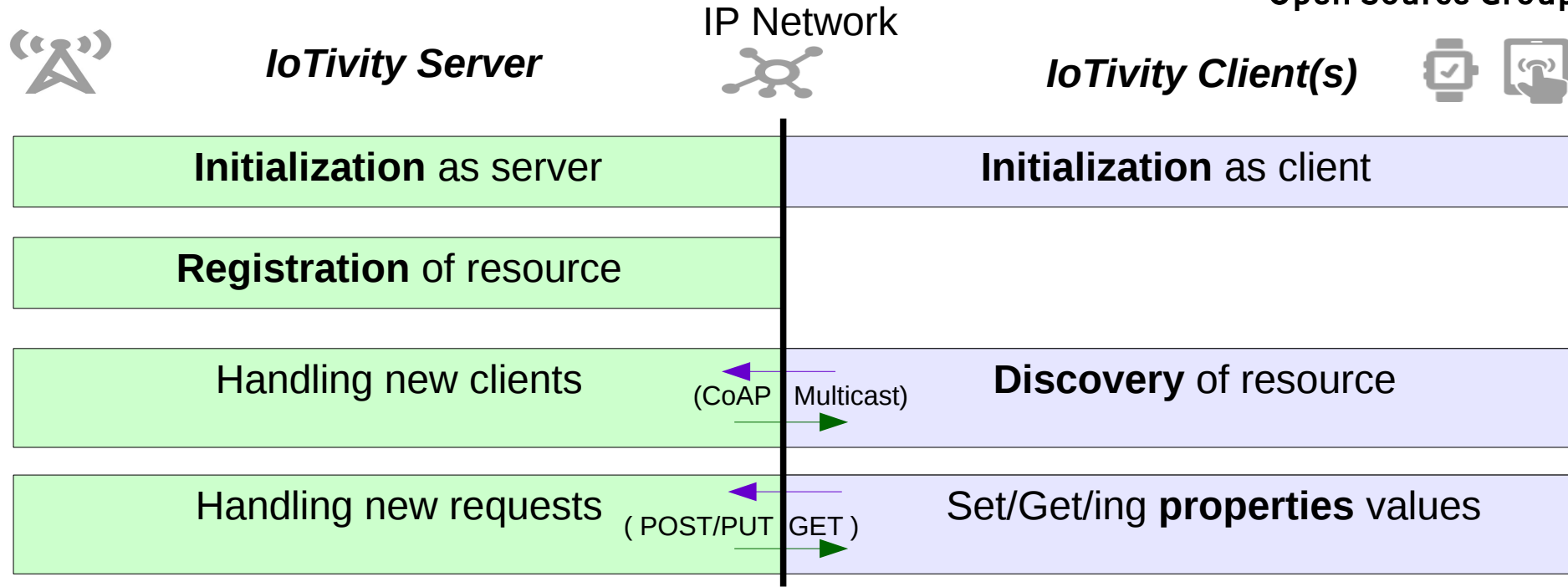


Get your hands on IoTivity!

- Get and build libraries: <https://wiki.iotivity.org/build>
 - Download sources and dependencies
 - Build it using **scons**
 - Or if OS shipping IoTivity (Tizen, Yocto, ...)
 - Use it a regular **library** (CPPFLAGS & LDFLAGS)
- Look at tree: <https://wiki.iotivity.org/sources>
 - Samples apps: resource/examples
 - C++ SDK: resource/resource/src
 - C SDK: resource/csdk



Typical flow



- Minimal example project to base on: git clone iotivity-example
 - Simple C (uses callbacks) or C++11



“Talk is cheap.
Show me **the code.**”
~ *Linus Torvalds*

Initialization

SAMSUNG

Open Source Group



IoTivity Server

IP Network



IoTivity Client(s)



```
OCPlatform::Configure(OC::PlatformConfig )
```

```
class IoTServer {  
    int main() { init(); ... }  
  
    OC::PlatformConfig mPlatformConfig;  
    void init() {  
        mPlatformConfig = OC::PlatformConfig  
            (OC::ServiceType::InProc,  
             OC::ModeType::Server, // different than C  
             "0.0.0.0", 0, // default for all subnets / ifaces  
             OC::QualityOfService::LowQos //or HighQos  
            );  
        OCPlatform::Configure(mPlatformConfig);  
    }  
};
```

```
OCPlatform::Configure(OC::PlatformConfig )
```

```
class IoTClient {  
    int main() { init(); ... }  
  
    OC::PlatformConfig mPlatformConfig;  
    void init() {  
        mPlatformConfig = OC::PlatformConfig  
            (OC::ServiceType::InProc,  
             OC::ModeType::Client, // different than S  
             "0.0.0.0", 0, // on any random port available  
             OC::QualityOfService::LowQos // or HighQos  
            );  
        OCPlatform::Configure(mPlatformConfig);  
    }  
};
```

Registration of resource on Server

SAMSUNG

Open Source Group



IoTivity Server

IP Network



IoTivity Client(s)



OCPlatform::Configure(PlatformConfig)
OCPlatform::registerResource(...)

OCPlatform::Configure(PlatformConfig)
OCPlatform::findResource(...)

```
class IoTServer { // (...)  
    OCResourceHandle mResource;  
    OC::EntityHandler mHandler; // for CRUDN operations  
    void setup() { // (...)  
        result = OCPlatform::registerResource(mResource, // handle for resource  
            "/BinaryRelayURI", // Resource Uri,  
            "oic.r.switch.binary", "oic.if.baseline" // Type & Interface (default)  
            mHandler // Callback to proceed GET/POST (explained later)  
            OC_DISCOVERABLE | OC_OBSERVABLE // resource flags  
        );  
        OCPlatform::bindTypeToResource(mResource, ... ); // optionally  
    } };
```

Resource discovery on client : finding

SAMSUNG

Open Source Group



IoTivity Server

IP Network



IoTivity Client(s)



```
OCPlatform::Configure(OC::PlatformConfig )
OCPlatform::registerResource(...)
    { OCPlatform internal }
```

```
OCPlatform::Configure(OC::PlatformConfig )
OCPlatform::findResource(OC::FindCallback)
    IoTClient::onFind(OCResource)
```

```
class IoTClient{ // ...
    OC::FindCallback mFindCallback;
    void onFind(shared_ptr<OCResource> resource);
    void setup() { //...
        mFindCallback = bind(&IoTClient::onFind, this, placeholders::_1); //C++11 std::bind
        OCPlatform::findResource("", // default
            "/oic/res", // CoAP endpoint, or resource based filtering for switches
            CT_ADAPTER_IP, // connectivityType can BT, BLE or other supported protocol
            mFindCallback, // to be called on Server response
            OC::QualityOfService::LowQos // or HighQos
        );
    }
};
```

Resource discovered on client



IoTivity Server

IP Network



IoTivity Client(s)



```
OCPlatform::Configure(OC::PlatformConfig )
OCPlatform::registerResource(...)
{ OCPlatform internal }
```

```
OCPlatform::Configure(OC::PlatformConfig )
OCPlatform::findResource(OC::FindCallback)
IoTClient::onFind(OCResource)
```

```
class Resource { OCResourceHandle mResourceHandle; }; // Our resource for CRUDN
```

```
class IoTClient { // (...)
    std::shared_ptr<Resource> mResource;
    void onFind(shared_ptr<OCResource> resource) {
        if ("/BinarySwitchURI" == resource->uri())
            mResource = make_shared<Resource>(resource);
    }
};
```

Resource discovering on client

SAMSUNG

Open Source Group



IoTivity Server

IP Network



IoTivity Client(s)



OCPlatform::registerResource(...)
IoTServer::**handleEntity**(OCResourceRequest

OCPlatform::findResource(...)
IoTClient::mResource->**post**()

```
void IoTServer::setup() { //...
    OC::EntityHandler handler = bind(&IoTServer::handleEntity, this, placeholders::_1);
    OCPlatform::registerResource( ... handler ... ); ...
}
IoTServer::handleEntity(shared_ptr<OCResourceRequest> request) {
    string requestType = request->getRequestType();
    if ( requestType == "POST" ) { handlePost() } else { ... }
    auto response = std::make_shared<OC::OCResourceResponse>(); //...
    OCPlatform::sendResponse(response);
}
void IoTServer::handlePost(...) {}
```

Resource representation

SAMSUNG

Open Source Group



IoTivity Server

IP Network



IoTivity Client(s)



```
OCPlatform::registerResource(...)
IoTServer::handleEntity(OCResourceRequest)
IoTServer::handlePost(OCResourceRequest)
```

```
OCPlatform::findResource(...)
IoTClient::mResource->post(false)
```

```
void Resource::post(bool value) {
    OCRepresentation rep; QueryParamsMap params;
    rep.setValue("value", value); // property
    mOCResource->post(rep, params, mPostCallback);
}
```

```
IoTServer::handlePost(shared_ptr<OCResourceRequest> request) {
    OCRepresentation requestRep = request->getResourceRepresentation();
    if (requestRep.hasAttribute("value")) {
        bool value = requestRep.getValue<bool>("value");
        cout << "value=" << value << endl; // OR set physical IO (GPIO...)
    }
}
```

GET / POST using Entity Handler

SAMSUNG

Open Source Group



IoTivity Server

IP Network



IoTivity Client(s)



```
OCPlatform::Configure(OC::PlatformConfig )
OCPlatform::registerResource(...)
```

```
OCPlatform::Configure(OC::PlatformConfig )
OCPlatform::findResource(...)
```

```
OC::EntityHandler(OCResourceRequest) {
  switch(getRequestType) {
    case 'POST: // Create resource 1st
      ...
    case 'GET' : // Retrieve current value
      ...
    case 'PUT' : // Not allowed for Switch
      ...
    OCPlatform::sendResponse(...);
    OCPlatform::notifyAllObservers();
  }
}
```

```
OC::OCResource::post(...) // Create
OC::PutCallback(...)
```

```
OC:ObserveCallback(...) // Notify
```

```
OC::OCResource::get(...) // Retrieve
OC::GetCallback(...)
```

“I'm not crazy. My **reality**
is just different from yours.”
~ *Lewis Carroll*

Resource is physical, not a boolean !

- General Purpose Input Output: **GPIO**
 - Set a voltage on electrical pin from userspace
- This can be set using Linux's **sysfs** (adapt to C/C++)
 - `echo $n > /sys/class/gpio/export ; echo out > /sys/class/gpio/gpio$n/direction`
 - `echo 1 ; sleep 1 ; echo 0 > /sys/class/gpio/gpio$gpio/value`
- Or faster with direct access (kernel registers...)
 - Even better using mapping library **MIRAA** (Along UPM for sensors drivers)
- So, server's “**entity handler**” should send signal on **POST/PUT** requests, that's all
 - `IoTServer::handleEntity() { ... IoTServer::handlePut() ... }`
 - `IoTServer::handlePut() { ~ write("/sys/class/gpio/gpio$n/value" , "%d", requestRep.getValue<bool>("value")); }`





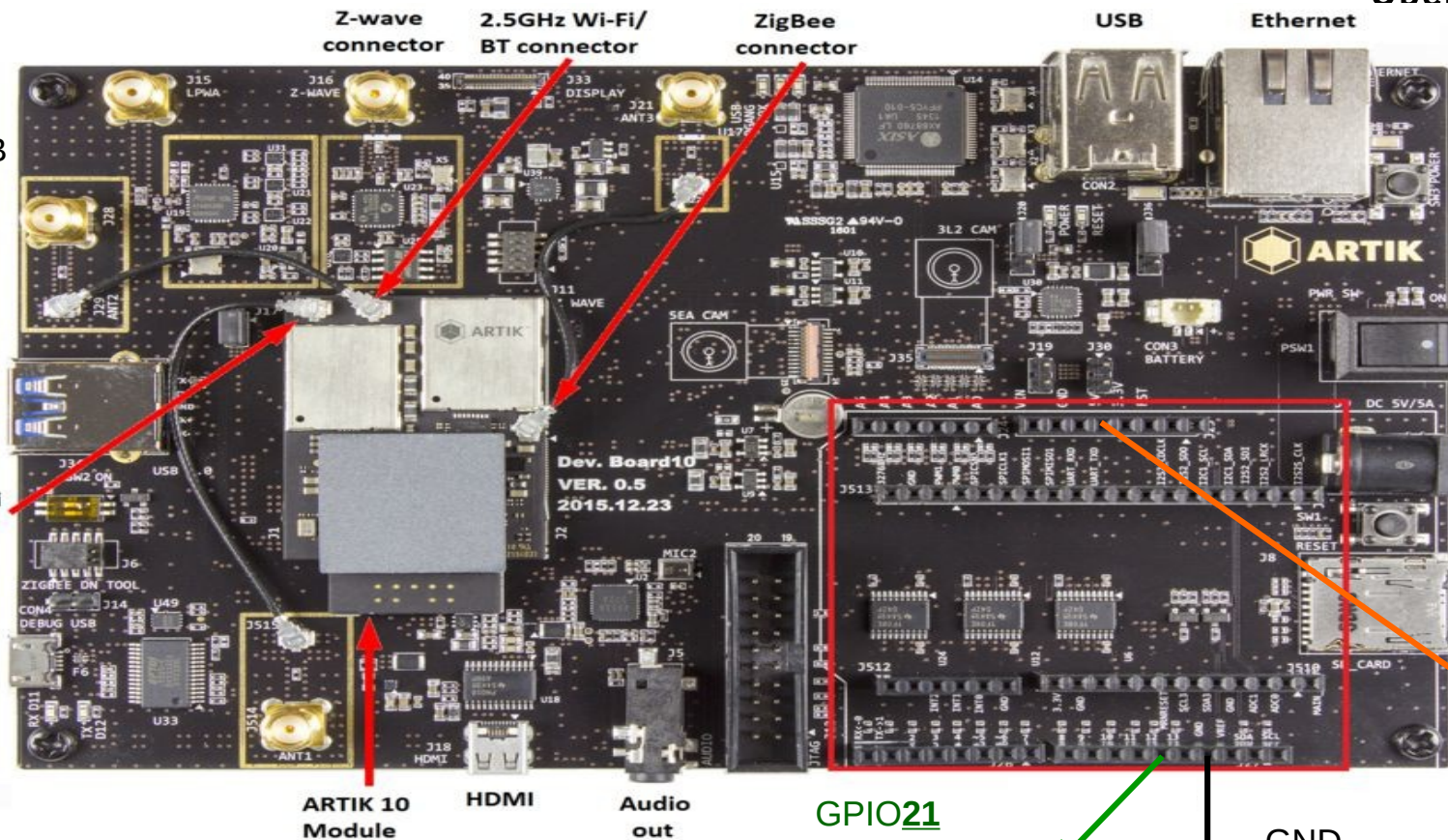
Samsung

ARTIK™ 10 GPIO pinout

SAMSUNG

Open Source Group

RAM:2GB
MMC:16GB



Power button

Power switch

5VDC input

Reset switch

microSD

Vcc1 +5V
@J15
(5th from right)

GPIO21
@J27/Pin12
(6th from right)

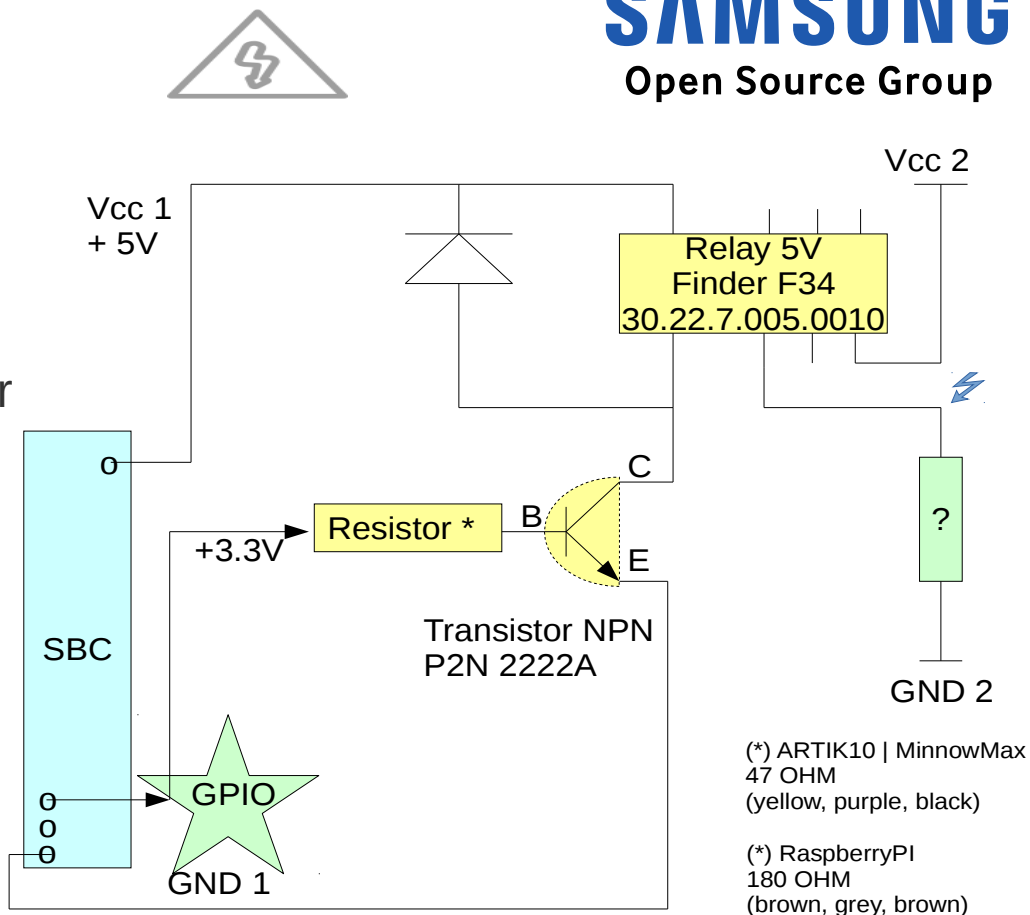
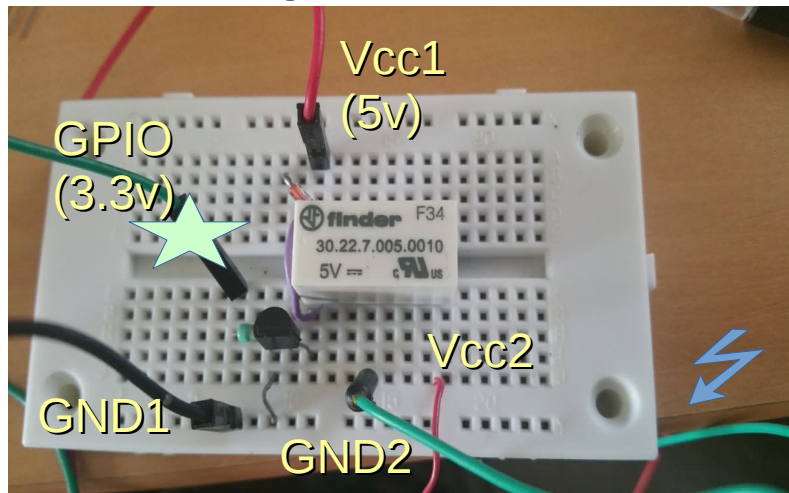
GND
@J27/Pin17
(4th from right)

CPU: Exynos5 (4+4 Cores)

Samsung Open Source Group

Hardware integration : DIY

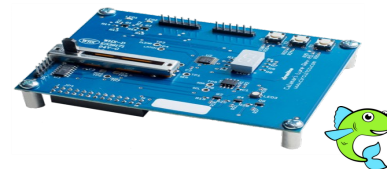
- High voltage relay (0-220V)
 - GPIO (3v3) < Relay (5V)
 - Signal Base of NPN Transistor



Samsung
ARTIK™

Hardware integration, with modules

- Simple modules, to wire on headers
 - I.e: Single channel Relay (HXJ-36)
- Daughters boards, (compatible headers)
 - Shields: for Arduino, and compatible SBC (ARTIK10, Atmel Xpl)
 - Hats for Raspberry Pi+ (RabbitMax ships relay, I2C, IR, LCD)
 - Lures for Minnowboard (Calamari has buttons, Tadpole transistors)
- Warning: Arduino Mega's GPIO is 5V and most SBC are 3.3V



IoT devices are constrained !

- If GNU/Linux is not an option for the computing power you have
 - Now let's port it to MCU using C
 - CSDK : [iotivity/resource/csdk](#)
 - Can use the same code base for Linux | Arduino...
- Example:
 - `git clone -b csdk iotivity-example`
 - `git clone -b arduino iotivity-example`
 - AVR binary Footprint : 116534 bytes for ATmega2560



IoTivity CSDK flow

SAMSUNG

Open Source Group



IoTivity Server

IP Network



IoTivity Client(s)



```
OCInit(NULL, 0, OC_SERVER);  
OCCreateResource( ..., handleOCEntity);  
    { OCProcess(); }
```

```
OCInit(NULL, 0, OC_CLIENT);  
OCDoResource(..., OC_REST_DISCOVER, ...)  
    handleDiscover(... OCClientResponse ...)
```








```
handleOCEntity(entityHandlerRequest) {  
    switch (entityHandlerRequest->method  
    {  
        case 'POST: // CREATE resource 1st  
        case 'GET' : // READ current value  
        case 'PUT' : // then UPDATE value  
        ...  
        OCDoResponse(&response);  
    }}
```

```
OCDoResource(...OC_REST_POST ...)  
    handlePost(... OCClientResponse ...)
```

```
OCDoResource(...OC_REST_GET ...)  
    handleGet(... OCClientResponse ...)
```

```
OCDoResource(...OC_REST_PUT ...)  
    handlePut(... OCClientResponse ...)
```


Interaction with other OS / Devices

- Consumer electronics products
 - Tizen ❤️ IoTivity
 - Tizen:3 contains as platform package (.rpm)
 - Tizen:2 can ship lib into native app (.tpk)
 - For Samsung Z1 (Tizen:2.4:Mobile)
 - Samsung GearS2 (Tizen:2.3.1:Wearable)
- GNU/Linux:    
 - Yocto (Poky, AGL, GENIVI, OstroOS)
- Other OS too:   



TIZEN ™

“Any sufficiently
advanced technology
is indistinguishable
from **magic.**”
~ *Arthur C. Clarke*

Demonstration: tizen-artik-20161010rzt

<https://vimeo.com/186286428#tizen-artik-20161010rzt>

SAMSUNG

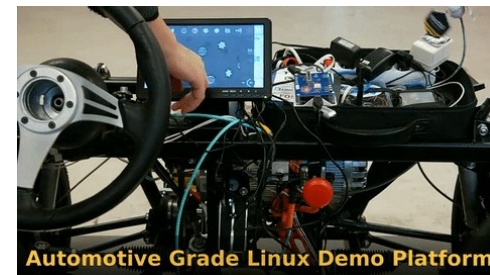
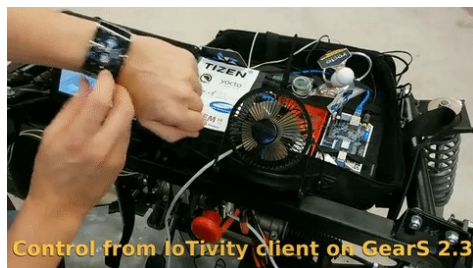
Open Source Group



Remote multi controlled binary switch
IoTivity Server on ARTIK10 (Tizen:3:Common)
connected to Tizen:2 clients with apps for:
Samsung Z1 & Samsung GearS2
using
IoTivity 1.2.0+RC3
<https://wiki.iotivity.org/tizen>
CC-BY-SA: <https://blogs.s-osg.org/author/pcoval/>

Want More ?

- More security, enable it, device provisioning, iotcon...
- More constrained: iotivity-constrained (RIOT, Contiki, Zephyr)
- More connectivity: BT, BLE, Zigbee, LTE, NFC...
- Scale: Deploy an OCF network of sensors, establish rules.
 - Global: Webservices (WSI), with cloud backend
 - For Smart (Home | Car | City | \$profile)



Conclusion

- Prototyping an IoT device is possible
 - with IoTivity IoT framework, that provides
 - Device to Device seamless connection
 - Create, Read, Update, Delete Resource & Notification
 - Can be easily implemented In C or C++
 - On Single Board Computers supporting Linux
- To work with devices supporting OCF standard protocol
 - Or supporting IoTivity like Tizen Wearables
- Possibilities are infinites



References

- Entry point:
 - <https://wiki.iotivity.org/examples>
- Technical references
 - <https://openconnectivity.org/resources/iotivity>
 - OIC_1.1_Candidate_Specification.zip
 - <https://wiki.iotivity.org/sources>
 - <http://elinux.org/ARTIK>
- Keep in touch online:
 - <https://wiki.iotivity.org/community>
 - <https://wiki.tizen.org/wiki/Meeting>
 - <https://developer.artik.io/forums/users/rzr>
 - <https://blogs.s-osg.org/author/pcoval/>



Danke Schoen ! Thanks / Merci / 고맙습니다

*Samung OSG, SSI,
Open Connectivity Foundation, LinuxFoundation,
FLOSS Communities: Tizen, Yocto, EFL, AGL, GENIVI
FlatIcons (CC BY 3.0) : Freepik, Chao@TelecomBretagne,
Libreoffice, openshot,
SRUK, SEF, Intel, Rabbitmax,
ELC/OpenIoT attendees,
YOU !*

Contact:
<https://wiki.tizen.org/wiki/User:Pcoval>

Q&A or/and Annexes ?

Demonstration: [iotivity-arduino-20161006rzt](https://vimeo.com/185851073#iotivity-arduino-20161006rzt)

<https://vimeo.com/185851073#iotivity-arduino-20161006rzt>

SAMSUNG

Open Source Group



IoTivity

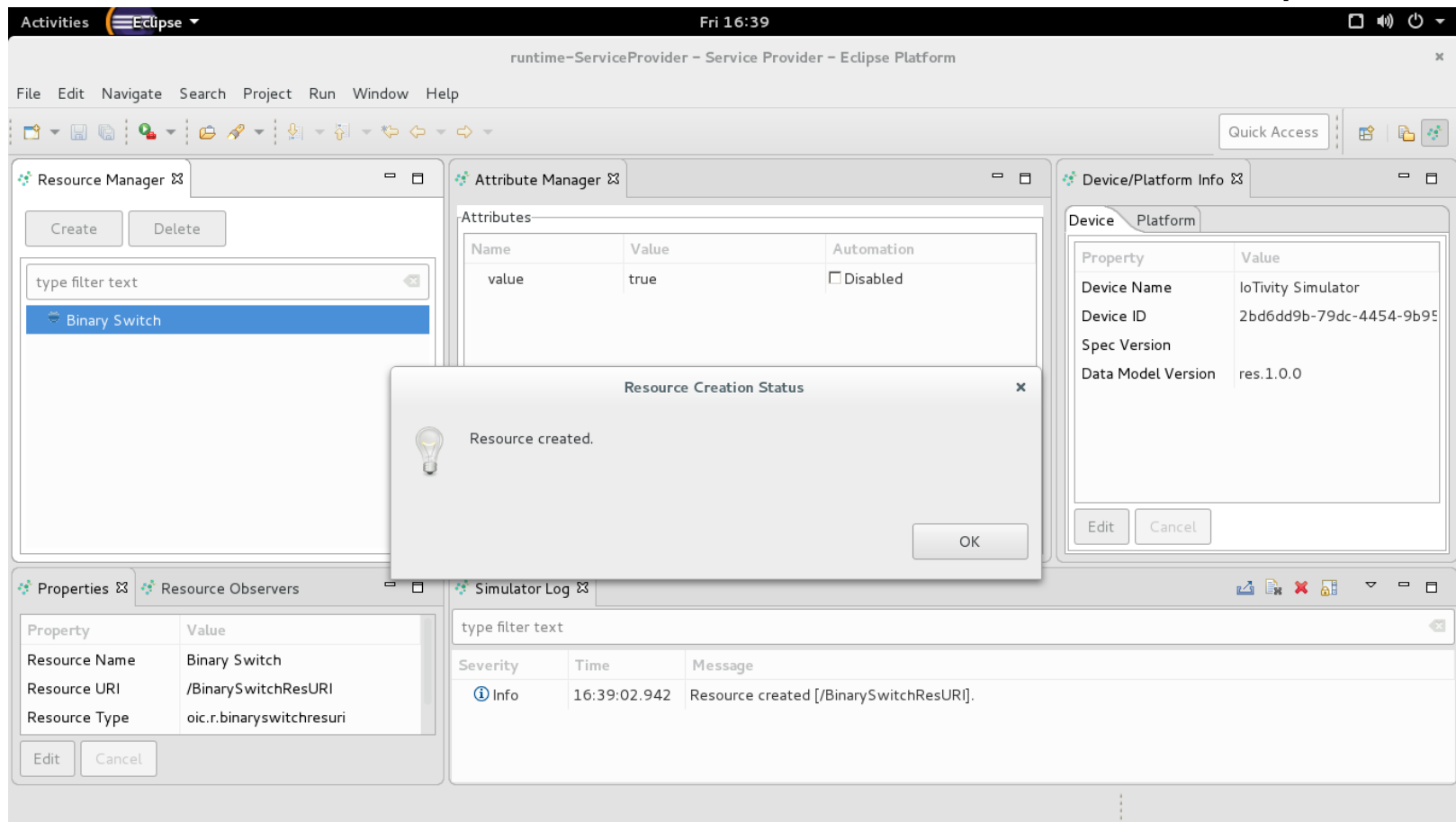
Binary Switch Example running on
ATmega2560 MCU (aka Arduino Mega + Eth Shield)
RaspberryPI 2 (with RabbitMax Hat)

Plus Tizen devices:

Samsung Z1 Mobile & Gear S2 Wearable

CC BY SA 3.0 : <https://blogs.s-osg.org/author/pcoval/>

Simulator: Importing model, Resource served



Client discovered Resource

The screenshot shows the Eclipse IDE interface with the following components:

- Activities:** runtime-ClientController - Client Controller - Eclipse Platform
- File Menu:** File, Edit, Navigate, Search, Project, Run, Window, Help
- Resource Manager:** Contains a search bar with "type filter text" and a list of found resources including "/BinarySwitchResURI".
- Attribute Manager:** Contains a table with columns "Attribute Name" and "Attribute Value".
- Properties:** Contains a table with columns "Property" and "Value".
- Simulator Log:** Contains a log of events.

The Simulator Log shows the following events:

Severity	Time	Message
Info	16:39:20.703	Resource Found [/BinarySwitchResURI].
Info	16:39:20.707	Sending GET request.
Info	16:39:20.739	Response received for GET.

Resources properties on client

The screenshot displays the Eclipse IDE interface for the `runtime-ClientController` project. The main window is titled `runtime-ClientController - Client Controller - Eclipse Platform`. The interface is divided into several panels:

- Resource Manager:** Contains a `Find Resources` button, a `Refresh` button, and a list of `Found Resources`. The list shows a single resource: `/BinarySwitchResURI`.
- Attribute Manager:** Displays a table of attributes with the following data:

Attribute Name	Attribute Value
value	true

Below the table are buttons for `GET`, `PUT`, `POST`, `Observe`, and `Automation`.
- Properties:** Shows a table of properties with the following data:

Property	Value
Resource URI	/BinarySwitchResURI
Address	coap://[fe80::20c:29ff:fe86:5c
Connectivity Type	SIMULATOR_CT_DEFAULT
Observable	Yes
Resource Types	oic.r.binaryswitchresuri
Resource Interfaces	oic.if.baseline, oic.if.a
- Simulator Log:** Displays a log of events with the following data:

Severity	Time	Message
Info	16:39:20.703	Resource Found [/BinarySwitchResURI].
Info	16:39:20.707	Sending GET request.
Info	16:39:20.739	Response received for GET.

Attempt to change property using PUT

The screenshot shows the Eclipse IDE interface. The main window is titled 'runtime-ClientController - Client Controller - Eclipse Platform'. The 'Resource Manager' on the left shows a list of resources with '/BinarySwitchResURI' selected. The 'Properties' view on the right shows the following data:

Property	Value
Resource URI	/BinarySwitchResURI
Address	coap://[fe80::20c:29ff:fe86:5c]
Connectivity Type	SIMULATOR_CT_DEFAULT
Observable	Yes
Resource Types	oic.r.binaryswitchresuri
Resource Interfaces	oic.if.baseline, oic.if.a

The 'Generate PUT Request' dialog is open in the center. It contains the following information:

Generate PUT Request
Dialog which takes input and generates a put request.

Interface Type: Baseline (oic.if.baseline)

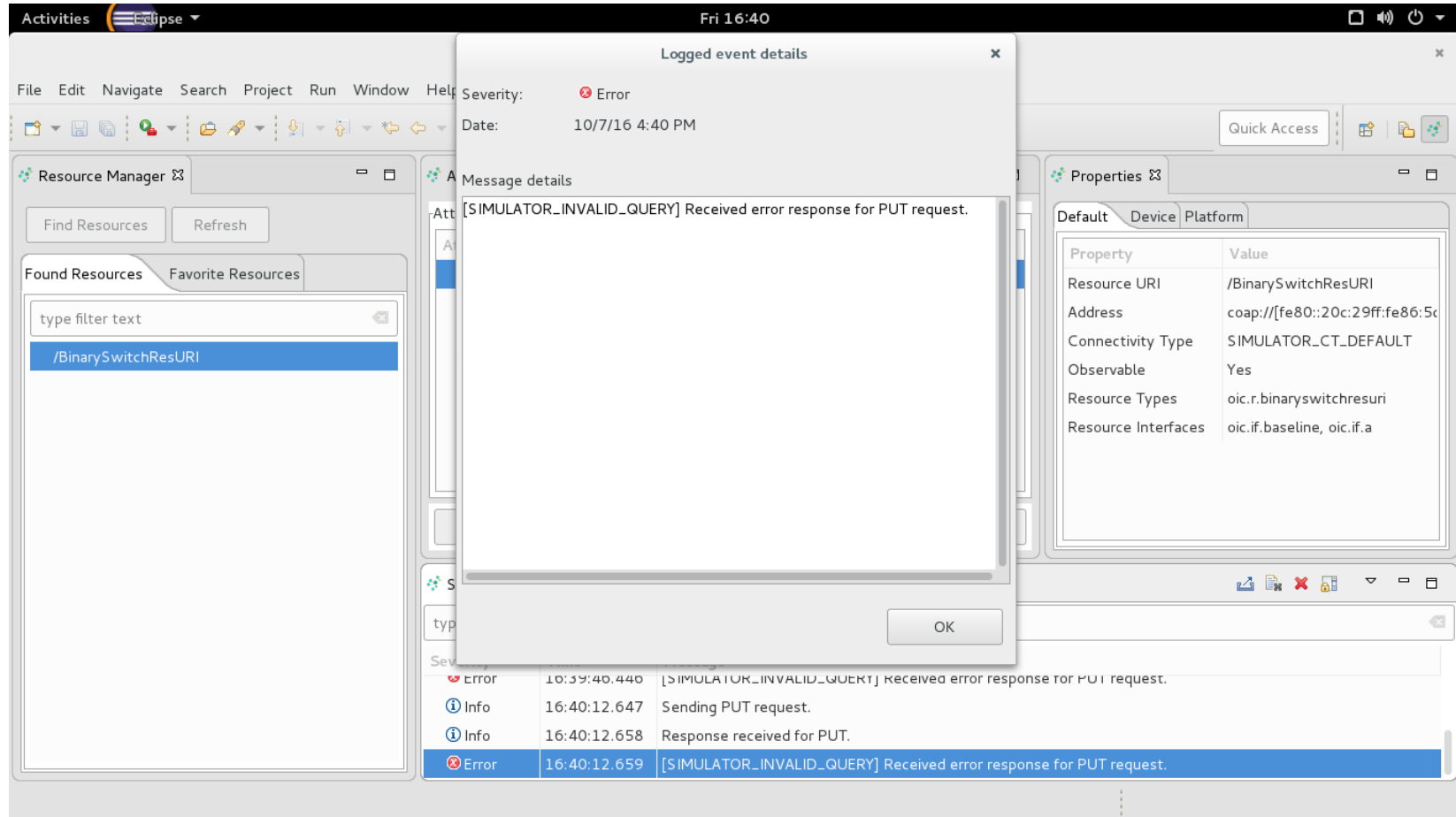
Name	Value
value	false

Buttons: Cancel, PUT

The 'Simulator Log' at the bottom shows the following messages:

Severity	Time	Message
Info	16:39:20.703	Resource Found [/BinarySwitchResURI].
Info	16:39:20.707	Sending GET request.
Info	16:39:20.739	Response received for GET.

Failed, as unsupported by interface from model



Client sets resource property using POST

The screenshot shows the Eclipse IDE interface for the 'runtime-ClientController' project. The main window is titled 'runtime-ClientController - Client Controller - Eclipse Platform'. The interface is divided into several panels:

- Resource Manager:** Contains 'Find Resources' and 'Refresh' buttons. Below, it shows 'Found Resources' and 'Favorite Resources' tabs. A search filter 'type filter text' is present, and a resource '/BinarySwitchResURI' is listed.
- Attribute Manager:** Contains an 'Attributes' table with columns 'Attribute Name' and 'Attribute Value'. The table shows a single entry: 'value' with a value of 'false'. Below the table are buttons for 'GET', 'PUT', 'POST', 'Observe', and 'Automation'.
- Properties:** Contains tabs for 'Default', 'Device', and 'Platform'. The 'Default' tab is active, showing a table of properties and values:

Property	Value
Resource URI	/BinarySwitchResURI
Address	coap://[fe80::20c:29ff:fe86:5c...
Connectivity Type	SIMULATOR_CT_DEFAULT
Observable	Yes
Resource Types	oic.r.binaryswitchresuri
Resource Interfaces	oic.if.baseline, oic.if.a
- Simulator Log:** Contains a 'type filter text' field and a table of log entries:

Severity	Time	Message
Info	16:40:12.050	response received for PUT.
Error	16:40:12.659	[SIMULATOR_INVALID_QUERY] Received error response for PUT request.
Info	16:40:31.378	Sending POST request.
Info	16:40:31.461	Response received for POST.

Server receives request and updates property

The screenshot displays the Eclipse IDE interface for the 'runtime-ServiceProvider' project. The main workspace is divided into several panels:

- Resource Manager:** Contains a 'Create' button, a 'Delete' button, and a list of resources. The 'Binary Switch' resource is selected.
- Attribute Manager:** Displays a table of attributes for the selected resource.
- Device/Platform Info:** Shows details about the device and platform.
- Properties:** Displays the properties of the selected resource.
- Resource Observers:** A panel for managing resource observers.
- Simulator Log:** A log window showing the sequence of events.

The **Attribute Manager** table shows the following data:

Name	Value	Automation
value	false	<input type="checkbox"/> Disabled

The **Device/Platform Info** panel shows the following data:

Property	Value
Device Name	IoTivity Simulator
Device ID	2bd6dd9b-79dc-4454-9b95
Spec Version	
Data Model Version	res.1.0.0

The **Properties** panel shows the following data:

Property	Value
Resource Name	Binary Switch
Resource URI	/BinarySwitchResURI
Resource Type	oic.r.binaryswitchresuri

The **Simulator Log** shows the following sequence of events:

Severity	Time	Message
Error	10:40:12.030	[/BinarySwitchResURI] Resource does not support PUT request!
Info	16:40:12.651	[/BinarySwitchResURI] Sent response for PUT request.
Info	16:40:31.383	[/BinarySwitchResURI] POST request received.
Info	16:40:31.448	[/BinarySwitchResURI] Sent response for POST request



Tizen devices connected with IoTivity

Phil Coval / Samsung OSG

What is demonstrated



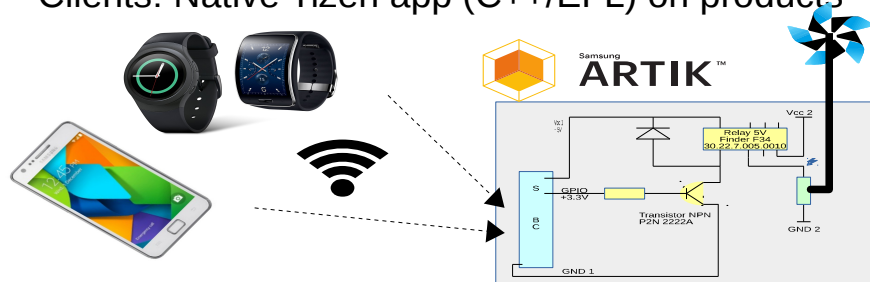
Seamless device to device connectivity framework



Linux-based software platform for consumer electronics

Demo: "Tizen DIY IoT Fan" controlled by devices

- Server: Tizen:Common on ARTIK dev board
- Clients: Native Tizen app (C++/EFL) on products



Hardware Information

ARTIK 10 (Exynos 5422 <http://elinux.org/ARTIK>)

Z1, Gear S (<https://wiki.tizen.org/wiki/Devices>)

Technical Showcase

CE Workgroup Linux Foundation / Embedded Linux Conference Europe

SAMSUNG

Open Source Group

What was improved

Tizen:Common 2016

- ARTIK 5 & 10 as latest reference devices
- Graphics: Enlightenment on Wayland 

IoTivity 1.2.0

- Notification service
- Cloud features
- OS Support (Windows)
- CoAP (TSL, HTTP)
- UPnP bridge
- Extending support:
 - New OS: Windows, macOS
 - Linux: Tizen, Yocto, Debian, WRT...
 - Hardware: x86, ARM, RPi, MCU (Arduino)

Yocto project efforts (meta-oic, meta-artik...) 

Source code or detail technical information availability

<https://wiki.tizen.org/wiki/User:Pcoval>

<https://wiki.iotivity.org/community>